Erick Cézar Oliveira Nascimento, Israel Barth Rubio,
Juliano Creppo Mendieta

Aumentando a escalabilidade de redes Openflow através de cache de mensagens do controlador

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Luis Carlos Erpen de Bona.

Curitiba PR
2016

Resumo

Redes Definidas por Software ainda possuem diversos problemas que impedem sua ampla utilização em ambientes reais, como por exemplo a escalabilidade de rede. A medida que o número de Switches aumenta é esperado que o número de requisições de regras que chegam ao(s) Controlador(es) também aumente e, com isso, cause uma negação de serviço na rede ou gere atrasos no envio das mensagens de resposta. Com o objetivo de trazer uma solução menos custosa computacionalmente para aumentar a escalabilidade em SDN's propusemos a introdução de um componente intermediário entre os Switches e o Controlador, sendo esse uma Cache de mensagens do Controlador com o objetivo reduzir o número de requisições de regras ao mesmo. Toda a experimentação foi realizada em ambiente virtual com a utilização do Mininet, contando apenas com um Controlador SDN e uma topologia de Switches e Caches customizável ligados ao mesmo, com a geração de um grande fluxo de requisições através do *pingall*, função nativa do Mininet. A aplicação utilizada para os testes foi um Firewall básico, cuja regra no Controlador SDN envia a resposta para que o Switch descarte o pacote cujo endereço esteja registrado no mesmo. Após os testes, notamos uma melhora no desempenho da rede com a implementação da Cache, em comparação com a mesma sem esse novo componente, diminuindo o número de requisições ao Controlador e apresentando uma melhora no tempo de resposta das mensagens. Desse modo, concluímos que a utilização de Caches das mensagens do Controlador pode ajudar a reduzir, mesmo que pouco, o problema de escalabilidade em SDN's.

Palavras-chave: SDN, OpenFlow, Escalabilidade, Cache, POX, OVS.

Abstract

Software Defined Networks still have several problems that prevent their wide use in real environments, such as the network scalability. As the number of Switches increases, it is expected that the number of rule requests arriving at the Controller will also increase and thus cause a denial of service on the network or generate delays in sending the response messages. In order to bring a computationally less costly solution to increase the scalability in SDN's, we proposed the introduction of an intermediate component between the Switches and the Controller, which is a Controller message Cache with the objective of reducing the number of rule requests to the Controller. All experimentation was performed in a virtual environment using Mininet, with only an SDN Controller and a customizable topology of Switches and Caches attached to it, with the generation of a large stream of requests through *pingall*, a native Mininet function. The application used for the tests was a basic Firewall, whose rule in the SDN Controller sends the response so that the Switch drops the packet whose address is registered in it. After the tests, we noticed a considerable improvement in the performance of the network with the implementation of the Cache, in comparison to the same without this new component, reducing the number of requests to the Controller and presenting an improvement in the response time of the messages. In this way, we conclude that the use of Caches of the messages of the Controller can help reduce, even if little, the problem of scalability in SDN's.

Keywords: SDN, OpenFlow, Scalability, Cache, POX, OVS.

Sumário

Lista de Figuras

Lista de Acrônimos

API	Application Program Interface
ARP	Address Resolution Protocol
CDPI	Control to Data-Plane Interface
CLI	Command Line Interface
IP	Internet Protocol
IT	Information Technology
KVM	Kernel-based Virtual Machine
MAC	Media Access Control
NBI	Northbound Interface
OVS	Open vSwitch
PPP	Point-to-Point Protocol
SBI	Southbound Interface
SDK	Software Development Kit
SDN	Software-Defined Network
T1	Teste 1
T2	Teste 2
T3	Teste 3
T4	Teste 4
TCP	Transmission Control Protocol
TMR	Tempo Médio de Resposta
TP1	Topologia 1
TP2	Topologia 2
TP3	Topologia 3
TP4	Topologia 4
TTL	Time to Live
VLAN	Virtual Local Area Network

Capítulo 1

Redes Definidas por Software - SDN's

A infraestrutura de rede atual utiliza o mesmo modelo há décadas, formado por vários Switches e Roteadores conectados por meios guiados como cabeamento e fibra ótica, e também meios não guiados como wireless. Deste modo, a cada alteração na rede (seja para desativar um determinado IP ou criar um caminho visando prover redundância) cada dispositivo ativo (Switches e Roteadores) deve ser configurado individualmente, eventualmente resultando em configurações erradas e erros como *forwarding loops*, perdas de pacotes e caminhos imprevistos. Outro problema da rede atual é a dificuldade, ou até mesmo a impossibilidade, de modificação dos softwares embarcados nos equipamentos fornecidos pela maioria dos fabricantes, visto que muitos deles são "caixas pretas"e só podem ser configurados através de Software Development Kits (SDK's) oferecidos por algumas empresas, permitindo certa flexibilidade, mas limitando a estrutura da rede a equipamentos do mesmo fornecedor e necessitando da instalação e configuração de novo firmware em todos os aparelhos individualmente. Até o início dos anos 2000, tinha-se como modelo padrão os Switches que concentravam tanto a tarefa de redirecionamento de pacotes (plano de dados), quanto as decisões de alto nível sobre os mesmos (plano de controle).

Rede Definida por Software (SDN - Software-Defined Network) é um modelo emergente utilizado para projetar, criar e gerenciar redes de computadores, e tem como objetivo superar as limitações de infra-estrutura das redes convencionais. Esse modelo permite que administradores de redes gerenciem os serviços da rede de uma maneira mais simples, com uma maior abstração da funcionalidade em baixo nível da rede, sendo feito através da virtualização, o que evita ter que configurar o hardware manualmente e torna possível que uma rede seja definida/modificada depois de ter sido implementada. O que possibilita essa abstração é a existência de um sistema Controlador, que permite ao software desenvolvido monitorar, definir e alterar a comutação da rede através de uma interface de programação bem definida. Estes Controladores de SDN podem ser considerados sistemas operacionais de redes de computadores, e assim como a maioria dos sistemas operacionais, esses são a ponte entre as aplicações que serão desenvolvidas para um maior controle da rede e hardware. Abstraindo as formas físicas da rede, o Controlador possibilita a criação de aplicações sob medida para cada rede específica e as necessidades geradas por ela.

Observando esses problemas, uma tecnologia chamada OpenFlow começou a ser desenvolvida, e teve sua versão inicial v1.0 publicada em 31 de Dezembro de 2009. A proposta do OpenFlow era dividir as tarefas de redirecionamento de pacotes e decisões de alto nível e criar uma interface padrão para poder manipular o Switch e permitir o desenvolvimento de novas funcionalidades. Abordaremos novamente esse assunto na seção 3.2

1.1 Arquitetura de uma SDN

SDN engloba múltiplos tipos de tecnologias de redes, projetadas para tornar a rede mais flexível e ágil, visando suportar servidores virtualizados e infraestruturas de armazenamento dos data centers modernos. Esse conceito surgiu originalmente para definir uma abordagem sobre como projetar, construir e gerenciar redes que separam os planos controle da rede (cérebro) e o encaminhamento (músculo), permitindo assim que o controle de rede se torne diretamente programável e que a infraestrutura adjacente seja abstraída para aplicações e serviços de rede. O funcionamento de uma SDN, na sua forma mais básica, centraliza o controle da rede separando a lógica do controle para um recurso computacional virtualizado. Todo modelo SDN possui alguma versão de um Controlador SDN, além de API's *southbound* e *northbound*.

O Controlador SDN atua como o cérebro da rede. Ele permite que usuários da SDN ganhem uma visão central de toda a rede, fazendo com que administradores de rede tenham mais poder para instruir Switches e Roteadores sobre como o plano de encaminhamento deve direcionar o tráfego de rede. APIs *southbound* enviam informações para os Switches e Roteadores da camada abaixo.

Considerando o padrão SDN, o OpenFlow foi a primeira API *southbound* e é um protocolo altamente utilizado. API's *northbound* enviam informações para a camada acima para aplicações e lógicas de negócio, dando ao administrador da rede a habilidade de controlar o tráfego da rede de maneira pragmática e iniciar serviços.

A Fig. 1.1 demonstra os componentes da arquitetura SDN a seguir:

- Aplicação SDN (SDN Application): São programas que, de maneira explicita, direta e programável, comunicam as necessidades da rede e o comportamento da rede para o Controlador SDN via Northbound Interfaces (NBI)'s. Uma Aplicação SDN consiste de uma lógica de Aplicação SDN e um ou mais NBI Drivers.

- Controlador SDN (SDN Controller): É uma entidade lógica centralizada com a função de traduzir as necessidades da Aplicação SDN para a camada inferior de plano de dados SDN e prover a Aplicação SDN uma visão abstrata da rede. Um Controlador SDN consiste de um ou mais agentes NBI, a lógica de controle SDN e o driver CDPI.

- Plano de Dados SDN (SDN Datapath): Dispositivo de rede lógica que inclui um Agente CDPI, um conjunto de um ou mais mecanismos de encaminhamento de tráfego e zero ou

mais funções de processamento de tráfego. Esses mecanismos e funções podem incluir um encaminhamento simples entre as interfaces externas de plano de dados ou processamento interno de trafego ou funções terminais.

- Interface Controladora do Plano de Dados SDN (SDN Control to Data-Plane Interface (CDPI)): Interface definida entre o Controlador SDN e o Plano de Dados SDN, provendo controle programático de todas as operações de encaminhamento, relato de estatísticas e notificações de eventos.

- SDN NBI: Interfaces entre Aplicações SDN e Controladores SDN, tipicamente proveem uma visões abstrata da rede e possibilitam expressões diretas do comportamento e necessidades da rede.

- Drivers de Interface& Agentes: Cada interface é implementada por um par driver-agente, o Agente representa a "*southern*", inferior ou lado voltado para a infraestrutura, e o driver representa o "*northern*", superior ou lado voltado a aplicação.

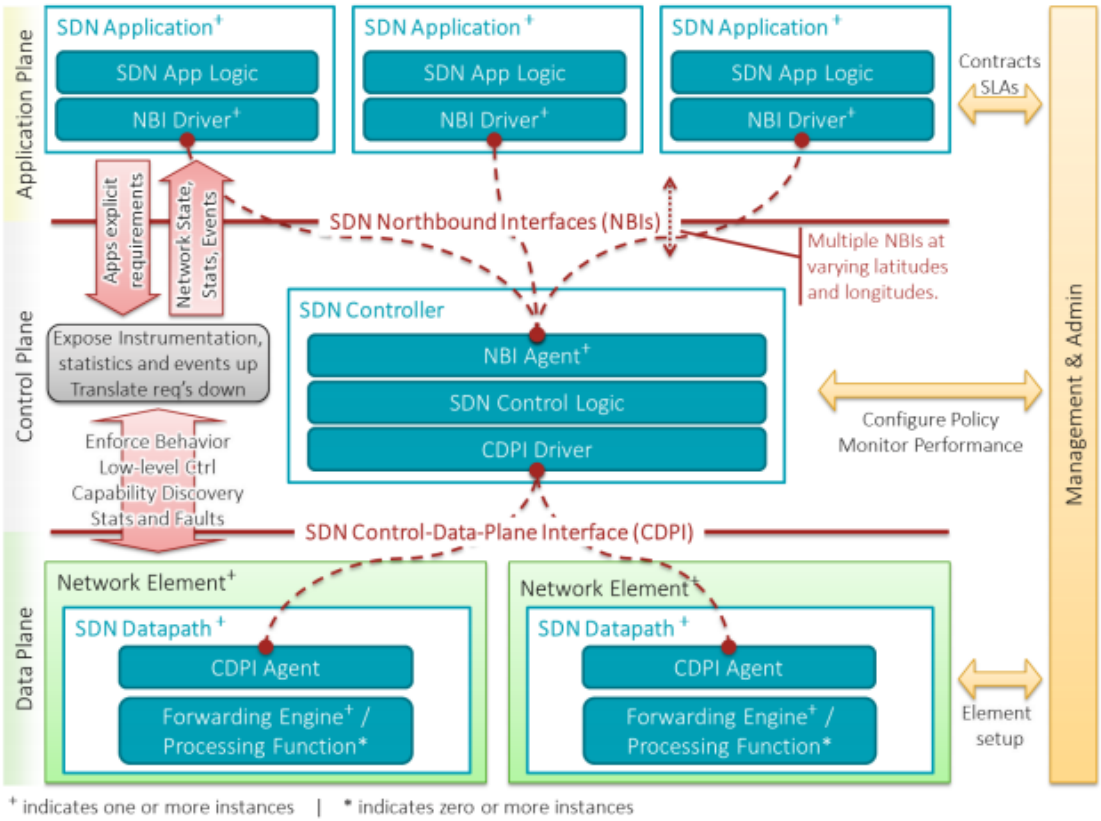

Figura 1.1: Visão geral da arquitetura SDN

1.2 Escalabilidade em SDN

SDN representa uma evolução no funcionamento das redes de computadores. A arquitetura SDN torna a rede mais simples, e abre espaço para a criação de novos serviços. O foco principal das SDN's são as aplicações realizadas no Controlador, e implementam serviços oferecidos na rede. O Controlador representa o cérebro da rede, já que as decisões quanto ao funcionamento da rede são de sua responsabilidade.

Uma possibilidade é uma rede com um único Controlador designado para controlar toda a funcionalidade de uma rede, como apresentado no artigo [16] e representado na Fig. 1.2. Dar total poder sobre uma rede para um único Controlador aparenta ser mais simples para desenvolver aplicações de controle, novas regras e politicas. Entretanto, o Controlador pode também se tornar o gargalo da rede. Quão maior a rede se torna, maior será a ocorrência de eventos e requisições enviadas ao Controlador, e desse modo, o Controlador pode não responder todas as requisições recebidas por sobrecarga de serviço.

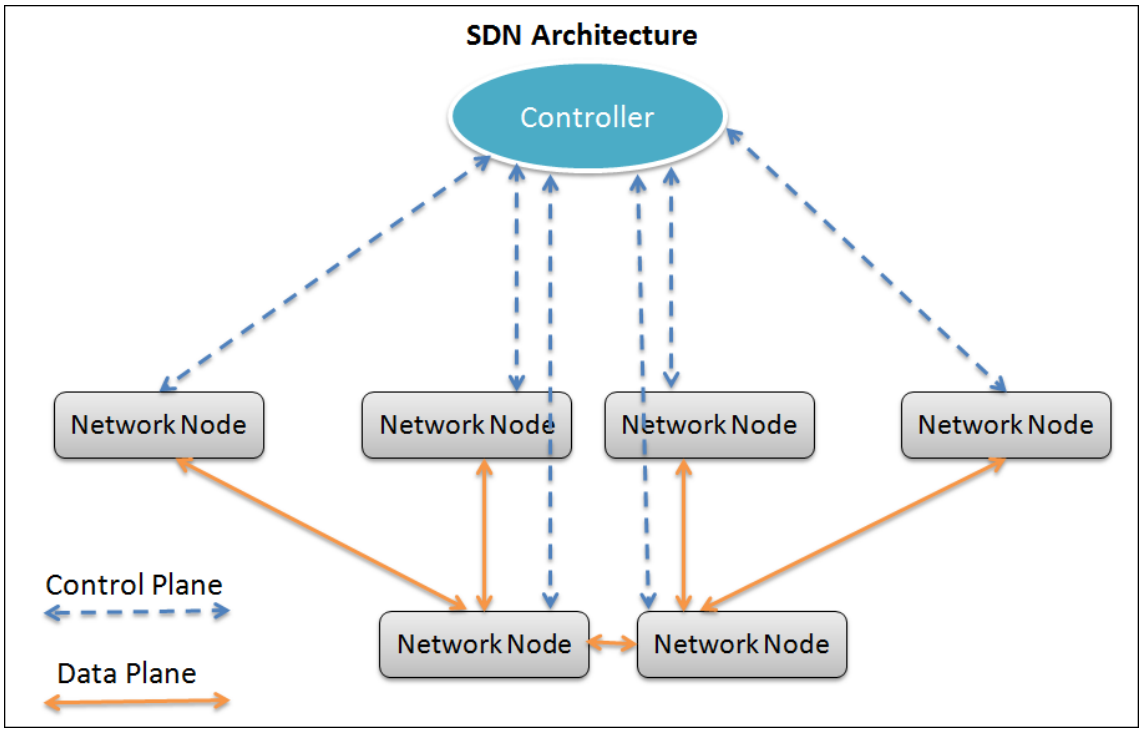

Figura 1.2: Arquitetura SDN

Um modo de aliviar esse problema, é a possibilidade de se trabalhar com multi Controladores funcionado paralelamente melhorando o desempenho da rede, ou de se trabalhar com Controladores multicamadas, como mostrado na Fig. 1.3.

Segundo [9], *"Escalabilidade é um dos desafios a ser enfrentado pela SDN"*. A escalabilidade é prejudicada tanto no diâmetro da rede, comutadores distantes podem não estar próximos de um Controlador, assim como no número de comutadores na rede, já que o número de Controladores pode não ser suficiente para responder em tempo hábil as requisições de todos comutadores. Segundo [13], *"A centralização física do controle gera um ponto único de falha"*.

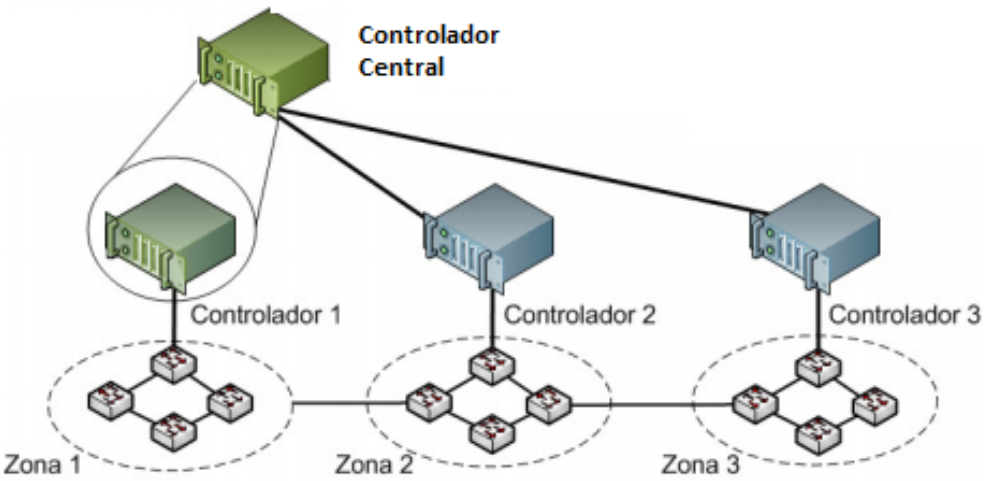

Figura 1.3: Arquitetura de multi-controladores

Tendo em vista a dificuldade encontrada na escalabilidade da rede, nosso trabalho propõe a implementação de Caches para mensagens do Controlador SDN, diminuindo assim o número de requisições que são redirecionadas ao Controlador, tendo como resultado o aumento da escalabilidade da rede.

1.3 OpenFlow

OpenFlow é um protocolo de comunicação utilizado na área de redes de computadores, que atua intermediando no plano de dados e plano de controle, redirecionamento de pacotes dos Switches, com o intuito de implementar o conceito conhecido como SDN.

Até o início dos anos 2000, tinha-se como modelo padrão os Switches que concentravam tanto a tarefa de redirecionamento de pacotes (plano de dados), quanto as decisões de alto nível sobre os mesmos (plano de controle). Além de realizar ambas as tarefas, os Switches detinham software e hardware proprietários, ou seja, cada fabricante "desenhava" o equipamento da maneira que lhe fosse mais conveniente.

As limitações do protocolo, principalmente a do software e hardware proprietários, dificultavam o desenvolvimento de novas funcionalidades aos Switches, pois elas seriam implementadas unicamente para uma determinada linha de equipamentos, tornando a implementação muito cara e sem muitos benefícios, uma vez que o público afetado seria muito restrito.

Observando os problemas da área de redes, tais como os equipamentos com software proprietário e de difícil manutenção, uma tecnologia chamada OpenFlow começou a ser desenvolvida, e teve sua versão inicial v1.0 publicada em 31 de Dezembro de 2009. A proposta do OpenFlow é dividir as tarefas de redirecionamento de pacotes e decisões de alto nível e criar uma interface padrão para poder manipular o Switch e permitir o desenvolvimento de novas funcionalidades. Segundo [11], *"redes programáveis precisam de Swicthes programáveis*

(usando virualização) que podem processar pacotes para múltiplos experimentos isolados nas redes simultaneamente".

Como levantado no parágrafo anterior, os Switches tem particularidades do fabricante, principalmente no que tange o software que o opera. No entanto, o OpenFlow tornou-se possível porque seus desenvolvedores exploraram algumas características em comum na tabela de redirecionamento de pacotes dos diversos modelos de Switch. Segundo [11], *"Enquanto a Flow Table de cada fabricante é diferente, nós identificamos um conjunto interessante de funções comuns que são executadas em muitos Switches e Roteadores. OpenFlow explora esse conjunto de funções comuns".*

O OpenFlow consiste basicamente de três componentes: Flow Tables, que contém ações para cada registro que case com suas regras; um canal seguro, que faz a conexão entre o Switch e o Controlador, permitindo envio de comandos e pacotes entre as partes; e por último, o próprio protocolo responsável pela comunicação entre os componentes. As ações mais simples são: redirecionar o pacote para uma determinada porta; encapsular o pacote e enviá-lo ao Controlador; e descartar o pacote, em geral por questões de segurança.

Cada entrada da Flow Table possui um header, que define o fluxo e serve para verificar o casamento de padrões com os pacotes que atravessam o Switch, uma ação correspondente à esse fluxo e estatísticas sobre o fluxo, tais como quantidade de vezes que foi utilizada e quantidade de pacotes/dados trafegados.

O funcionamento do fluxo de pacotes via protocolo OpenFlow, explicado em poucas palavras, é: os pacotes chegam no Switch e há então uma tentativa de encontrar regras que se apliquem à ele, ou seja, encontrar seu fluxo; Se o fluxo for encontrado, toma-se a ação descrita por ele, caso contrário, o pacote é enviado ao controlador, que geralmente encontra-se em um servidor comum; O Controlador então pode decidir entre criar uma nova entrada na Flow Table do Switch ou simplesmente processar o pacote, por exemplo.

O protocolo OpenFlow em si pode ser dividido em quatro partes: camada de mensagens, máquina de estados, interface de sistema e configurações. A camada de mensagens verifica se a estrutura e semântica dos pacotes estão corretas, e tem funcionalidades para criar, copiar, comparar, imprimir e manipular mensagens. A máquina de estados realiza as principais atividades de baixo nível, tais como verificação de capacidade de transmissão e controle de fluxo. A interface de sistema define como é a comunicação do protocolo com os diversos equipamentos. A parte de configurações define diversos parâmetros utilizados pelo protocolo OpenFlow.

O ciclo de vida de pacotes no Switch OpenFlow é definido como a seguir: o pacote chega por uma das portas e uma estrutura de meta-dados conhecida por chave é acoplada à eles. Essa chave contém informações sobre o buffer, cabeçalhos e portas utilizadas, e é utilizada para descobrir em qual das Flow Tables o pacote se encaixa, e no interior da tabela, para selecionar o fluxo correspondente. Se o fluxo for encontrado, suas ações são tomadas, caso contrário o pacote é enviado ao Controlador para uma análise mais elaborada.

Até o momento foram descritos o funcionamento e componentes do protocolo OpenFlow. Nas seções seguintes do presente capítulo, serão abordadas algumas características principais de cada versão lançada do protocolo, e haverá uma breve descrição sobre emuladores, exemplificando-os a partir do emulador Mininet.

1.3.1 Versões do protocolo OpenFlow

O protocolo OpenFlow teve sua primeira primeira versão lançada no ano de 2009 e encontra-se em constante evolução, contando com modificações e melhorias até os dias atuais (ano de 2016). Segundo [11], *"planos para expansão de funcionalidades são ambiciosos (e custosos), e levarão anos para serem desenvolvidos"*. Na sequência, serão apresentadas as principais características de cada versão do protocolo.

A versão 1.0 do protocolo OpenFlow foi lançada em 31 de Dezembro de 2009. Essa versão garantiu uma taxa mínima na banda de comunicação do protocolo, suporte a cookies na manutenção dos registros da Flow Table, casamento de padrão com pacotes ARP e uma nova precisão no controle da duração das mensagens de status e expiração.

A especificação da versão 1.1, de 28 de Fevereiro de 2011, trouxe consigo o suporte a diversas Flow Tables simultâneas, expandindo a capacidade do Controlador e melhorando o gerenciamento das regras, além de trazer o conceito de grupos, permitindo que um conjunto de portas possa ser reconhecido como uma única entidade, cuja qual pode ter uma tratativa padrão antes de fazer o redirecionamento dos pacotes. Introduziu também o conceito de porta virtual, deixando de contar apenas com as portas físicas do Switch.

Na versão 1.2, com data de 5 de Dezembro de 2011, houve uma melhoria na funcionalidade de casamento de padrões de acordo com os campos especificados. Para atender isso, o tamanho das estruturas de dados utilizadas teve que ficar mais flexível. Permitiu também a customização de mensagens de erro e adicionou suporte ao protocolo IPv6.

Em 13 de Abril de 2012 foi lançada a versão 1.3 do OpenFlow. Ela diminuiu a limitação das versões anteriores no que diz respeito à expressão das capacidades dos Switches OpenFlow utilizados. Contou também com uma simplificação nos registros da Flow Table para tratar casos de "table miss". Extendeu o suporte no IPv6, conseguindo interpretar os headers de tamanho variável contidos nesse protocolo, além de adicionar métricas aos fluxos da tabela, permitindo mensurar quantidade de dados trafegados de acordo com determinada regra. Adicionou também o suporte à conexões auxiliares, de forma que elas possam ajudar a conexão principal entre o Switch e o Controlador.

A versão 1.4 teve seu laçamento datado em 5 de Agosto de 2013. Trouxe um novo conjunto de portas ópticas, permitindo configurar e monitorar a transmissão e recebimento de dados por meio óptico. Houve também uma extensão na funcionalidade de monitoramento, permitindo ao administrador definir uma quantidade de monitores e customizá-los para verificar pacotes que atendessem uma determinada condição. Exterminou também a ocorrência de erros

por Flow Table cheia ao tentar inserir novos fluxos, deletando os fluxos de "menor importância". Adicionou alguns códigos de erro para tratar determinadas exceções.

Na versão 1.5, lançada em Dezembro de 2014, surgiu o conceito de "Egress Table". Com isso, o processamento deixou de ser feito completamente na porta de entrada, permitindo que ele fosse feito também na porta de saída. Extendeu também o suporte a outros protocolos, tais como IP e PPP, o que anteriormente não era possível, pois tratava somente Ethernet. Houve também uma expansão na funcionalidade de levantamento de estatísticas dos fluxos. Para diminuir o overhead causado por esse levantamento de estatísticas, também foi desenvolvida uma trigger para melhor gerenciar a funcionalidade. Foi incluído também o casamento de padrão com flags dos segmentos TCP na Flow Table. Com essa versão, também tornou-se possível o reconhecimento do status de cada Controlador presente na rede OpenFlow.

A versão estável atual do protocolo é a versão 1.5, porém novas melhorias estão sendo constantemente desenvolvidas. Ainda não há previsão para o lançamento da próxima versão, que irá englobar tais melhorias, estas que ainda não foram publicadas.

1.4 Emuladores

A experimentação de novos protocolos é difícil na área de redes, uma vez que ela já está bem engessada. Tais experimentos são feitos na própria rede e não podem comprometer o funcionamento da mesma. Segundo [11], *"A alta complexidade tornou as redes mais estáticas já que a IT procura evitar interrupção nos serviços"*. Estudos e pesquisas em redes não podem afetar o funcionamento normal da infra-estrutura. Visando cobrir essa necessidade, no que tange à SDN e ao OpenFlow, foram desenvolvidos emuladores.

Esse tipo de software (de emulação) traz benefícios ao campo do estudo e da pesquisa em redes de computadores, a um baixo custo computacional. Eles emulam uma rede real, permitindo que professores possam apresentar conceitos abstratos de forma mais acessível, e que pesquisadores possam fazer testes e experimentos na rede, sem afetar os usuários. Segundo [11], *"SDN motiva pesquisadores a testar as inovações deles em cenários com tráfego em tempo real, o que seria inviável se não existisse a tecnologia SDN"*.

Existem diversos emuladores de redes, entre eles: Network Simulator 3, Estinet e Mininet. Neste trabalho, focaremos o estudo no Mininet, pois é um emulador bem difundido atualmente, contando com uma boa documentação e vários exemplos de uso. Na próxima seção há uma breve descrição do mesmo.

Entretanto, ao invés de usar o Switch padrão do Mininet, que embora seja facilmente programável e modificável, utilizaremos o Open vSwitch (OVS) por utilizar a memória kernel, ao invés da memória de usuário como o Switch padrão do Mininet, tornando o tráfego da rede mais rápido. Nas próximas seções, haverá uma breve explicação sobre o OVS. DN

1.4.1 Mininet

Mininet é um emulador de redes de computadores que contém suporte à SDN, mais especificamente por meio de simulações de uma rede OpenFlow. Ele permite uma rápida prototipação de uma grande rede virtual OpenFlow, composta por Controlador, Switch, Hosts e Links.

O Mininet é um software versátil. Conta com topologias de rede pré-definidas, desde as mais simples, com apenas um Switch, até as mais complexas, com diversos Switchs. Além disso, há a opção de customizar a rede de acordo com as necessidades do usuário.

Esse emulador torna possível a criação de toda a rede virtual em apenas uma máquina, seja ela virtual ou física, e suportando até 4096 Switches virtuais OpenFlow ao mesmo tempo. Dispensa a necessidade do computador que hospeda o aplicativo estar conectada à rede física.

O Mininet permite que diversos desenvolvedores trabalhem concorrentemente, na mesma topologia, além do empacotamento de testes, e disponibilidade de uma CLI (Command Line Interface) para depuração compatível com as topologias e com o OpenFlow.

Os Switches e Hosts virtualizados funcionam da mesma forma que os dispositivos reais. É possível enviar dados entre os Hosts e analisar os fluxos contidos na tabela do plano de dados dos Switches componentes.

O Mininet conta com facilitadores para identificar o comportamento do sistema, além de executar código nativo Unix/Linux e ter conexão com aplicativos de mensuração de desempenho, tal como o iperf. Assim, o código desenvolvido e testado nos Controladores, Switches e Hosts do Mininet, é facilmente adaptado aos dispositivos físicos, com pequenos ajustes nas configurações.

1.4.2 Open vSwitch - OVS

Para melhor desempenho da solução proposta, utilizaremos o Open vSwitch. O OVS é uma implementação open-source de um Switch virtual. Em maio de 2010, a primeira versão do OVS foi disponibilizada, oferecendo suporte para um grande número de recursos. O Open vSwitch suporta as principais soluções de hypervisor (Gerenciador de recursos de hardware de uma máquina virtual) de software livre, incluindo o Kernel-based Virtual Machine (KVM), o VirtualBox , o Xen e o Xenserver. É um substituto para o módulo atual de ponte do Linux, utilizado como padrão no Mininet.

O OVS consiste em um "daemon"de comutador e um módulo do kernel concomitante que gerencia a comutação baseada no fluxo, por esse motivo escolhemos como o Switch padrão a proposta. É possível executar o Open vSwitch integralmente na memória do usuário, porém isso ocasiona a redução do desempenho.

1.5 Conclusão

Nesse capítulo abordamos SDN's, que são definidas através de elementos de rede, como Roteadores, Switchs, entre outros componentes de rede. Adotaremos SDN como padrão, pois requer um Controlador que é usado para se comunicar com Switchs físicos e virtuais, se encaixando perfeitamente com nossa proposta, que será melhor explicada nos próximos capítulos, de criar Caches para diminuir a carga sobre o Controlador. Como trabalharemos em um ambiente virtual, o Mininet foi escolhido por apresentar uma ampla comunidade além de possuir a capacidade de personalização na criação de topologias. Por fim apresentamos o Open vSwitch, um comutador virtual que trabalha diretamente com a memória kernel, sendo esse o motivo de escolha como padrão, melhorando o desempenho da rede virtual apesar de uma configuração mais complexa.

Capítulo 2

Proposta

Um dos possíveis projetos SDN é colocar toda a funcionalidade de controle em um Controlador centralizado, tornando o desenvolvimento de aplicações de controle e de políticas muito mais fáceis, visto que o Controlador tem a visão completa da topologia da rede. Com o aumento do tamanho da rede, cada vez mais eventos e requisições são enviados ao Controlador, fazendo com que, em algum ponto, o Controlador não consiga lidar com todas as requisições enviadas [16].

Em uma SDN que utiliza o protocolo OpenFlow de maneira reativa, sempre que um pacote com características específicas que ainda não possui regra na Flow Table chega ao Switch é necessário fazer a requisição dessa ao Controlador SDN [16]. Desse modo, numa rede onde n Switches estão conectados a um Controlador SDN central serão feitas, no pior caso onde o fluxo de pacotes percorrerá todos os Switches da rede, n requisições ao Controlador para o mesmo pacote visando preencher todas as Flow Tables.

Nesse trabalho de conclusão de curso, nós propomos a implementação de uma aplicação virtual de Caches para mensagens do controlador SDN, diminuindo assim o número de requisições que são redirecionadas ao Controlador, tendo como resultado o aumento da escalabilidade da rede. O objetivo da implementação é prover uma resposta mais rápida para a solicitação de regra do Switch a um pacote específico, uma vez que essa já tenha sido estabelecida anteriormente por requisição de algum outro Switch da rede. Os Switches devem enxergar as Caches como sendo o Controlador SDN, visando manter a estrutura da rede intacta na visão de cada Switch. A Fig. 5.1 demonstra a arquitetura de uma SDN definida de acordo com o contexto descrito acima. No exemplo, os cinco Switches pertencem ao fluxo e nenhum deles possui a regra correspondente ao pacote. Somente o primeiro Switch terá de aguardar o tempo de resposta do Controlador. Os demais aproveitarão a regra armazenada em Cache, aguardando assim, somente o tempo de resposta da Cache.

Os experimentos relativos a implementação serão feitos em ambiente virtual, utilizando o Mininet e o OVS para emular uma SDN que utiliza o protocolo OpenFlow, sendo essa composta apenas de um Controlador SDN central ligado a X Switches e Y Caches. Para a comparação dos resultados obtidos nos testes nesse ambiente descrito, criaremos um segundo ambiente com as

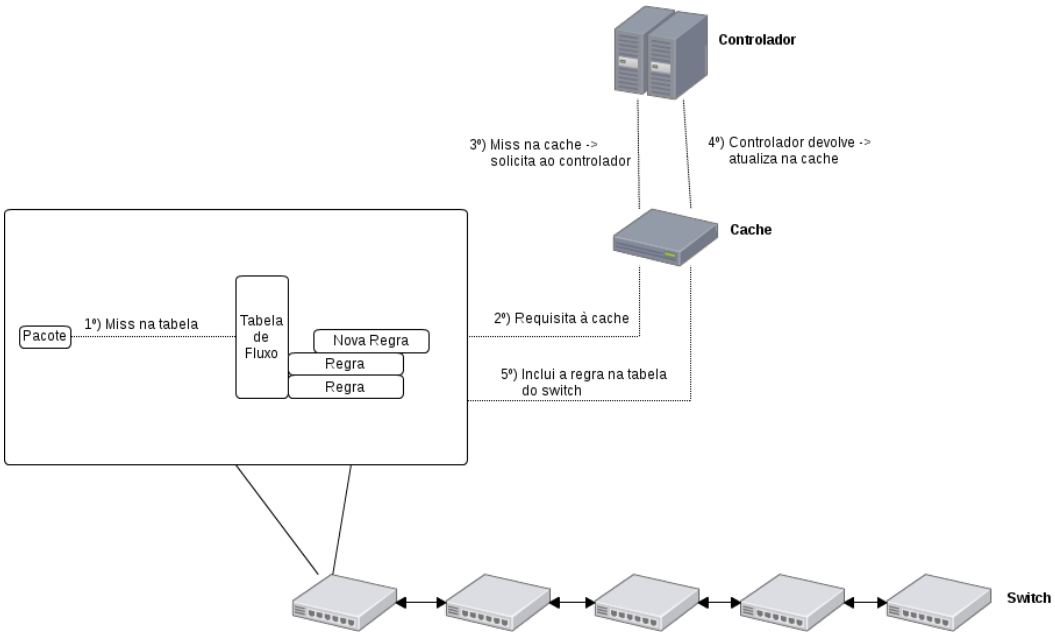

Figura 2.1: SDN com cache

mesmas características descritas acima, com exceção da Cache e as funcionalidades relativas a ela.

A utilização do Mininet como emulador da rede SDN se deve a sua grande popularidade na área e grande quantidade de documentação e exemplos de uso da ferramenta. Sobre a utilização do OVS, sua escolha está atrelada ao suporte que oferece a criação de Switches virtuais que suportam o protocolo OpenFlow, além disso, outra vantagem do OVS é a utilização da memória kernel ao invés da memória de usuário, utilizada pelo Switch padrão do Mininet, fornecendo um melhor desempenho na comutação de pacotes pois não há a necessidade dos pacotes precisarem cruzar o espaço de memória de usuário para o espaço do kernel e voltar a cada iteração. Enquanto o Switch no modo kernel os pacotes permanecem o tempo inteiro no espeço de memória kernel enquanto atravessam o Switch.

O Controlador SDN escolhido para a implementação foi o POX, cuja implementação é em Python. Sua escolha se deve a popularidade da ferramenta nas áreas de ensino e pesquisa, além de oferecer suporte para a utilização do OVS. Como consequência dessa escolha, nos limitamos a utilizar o protocolo Openflow 1.0, que é o mais recente suportado pelo Controlador.

Capítulo 3

Implementação

A Cache de mensagens do Controlador foi implementada com o objetivo de reduzir o número de requisições de novos Flows ao Controlador SDN em redes que possuem apenas um Controlador. Em uma rede com um número muito grande de Switchs pode ocorrer uma sobrecarga de mensagens PacketIn no Controlador, causando uma negação de serviço na rede ou gerar atrasos no envio das mensagens de resposta, caso a rede possua somente um Controlador. As requisições de novos Flows ocorrem basicamente em dois passos: Switch envia uma mensagem PacketIn solicitando regra para determinado pacote e o Controlador responde com uma mensagem Flow Mod para instalação/modificação/deleção da uma regra.

3.1 Mensagens PacketIn e Flow Mod

Uma mensagem com o pacote PacketIn é a forma com que os Switchs se comunicam com o Controlador. Há duas razões para está comunicação ocorrer: Após o recebimento de um pacote e a ocorrência de um miss na Flow Table ou quando ocorre match e a ação deve ser o encaminhamento para o controlador.

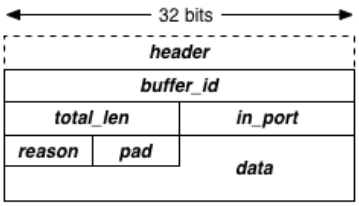

Figura 3.1: Estrutura PacketIn

Uma mensagem PacketIn consiste do *header*, que contém dados como versão, o tamanho do pacote (que indica o final do pacote em uma sequência de bytes) e um *xid* (um valor único usado para identificar match entre requisições e respostas). Em sequência há o *buffer id*, que é um valor único usado para identificar um pacote armazenado no buffer do Switch, de modo que seja possível atribuir a resposta do controlador ao pacote correto no Switch. O tamanho do pacote

capturado é indicado no campo *total len*. A porta do Switch por onde o pacote foi recebido é indicado no *in port*. O campo *reason* indica o motivo pelo qual o pacote foi encaminhado para o controlador. Finalmente, a parte capturada do pacote começa logo depois do campo *pad*. O campo *data* consiste da cópia do pacote que o Switch recebeu, não soube como tratar e veio a causar a criação de um PacketIn. Geralmente é um pacote Ethernet, que encapsula outros tipos de pacotes, como por exemplo um pacote IPV4, que estará contido no Ethernet.Payload da mensagem PacketIn. Esse campo de dados será analisado pelo Controlador, de forma a tentar encontrar algum match com as regras dele, e por consequência gerar algum tipo de resposta ao Switch solicitante, por meio de um PacketOut ou FlowMod, por exemplo.

Quando um Switch não encontra na sua Flow Table uma regra para um determinado pacote, o Switch em questão enviará uma mensagem PacketIn informando a razão para o envio do pacote. Há dois tipos de *reason* no OpenFlow 1.0:

```
/* Why is this packet being sent to the controller? */
enum ofp_packet_in_reason {
    OFPR_NO_MATCH,          /* No matching flow. */
    OFPR_ACTION             /* Action explicitly output to controller. */
};
```

Figura 3.2: Reason PacketIn

Ao receber um PacketIn, com *Flow miss* como *reason*, o Controlador prepara uma mensagem Flow Mod, para adição de um novo Flow para ser enviada ao Switch. A mensagem Flow Mod permite ao controlador modificar estado de um Switch OpenFlow, pois a responsabilidade da mensagem é modificar, adicionar ou deletar regras contidas na Flow Table de um Switch, [1].

```
enum ofp_flow_mod_command {
    OFPFC_ADD,              /* New flow. */
    OFPFC_MODIFY,           /* Modify all matching flows. */
    OFPFC_MODIFY_STRICT,    /* Modify entry strictly matching wildcards */
    OFPFC_DELETE,           /* Delete all matching flows. */
    OFPFC_DELETE_STRICT    /* Strictly match wildcards and priority. */
};
```

Figura 3.3: Comandos Flow Mod

Mais detalhes sobre **DELETE STRICT** e **MODIFY STRICT** encontram-se em [1].

As mensagens Flow Mod também contém o campo *Actions*. As *Actions* funcionam da seguinte forma: Quando ocorre um match de pacote no Switch, a regra contida na Flow Table indica uma ação para aquele pacote, tais como porta para qual o pacote deve ser redirecionado, *set* de IP(s) de destino ou origem, *set* de endereço MAC, *set* de VLAN ID, prioridade do pacote ou caso o campo *action* esteja vazio, é considerado *drop* de pacotes. Para mais detalhes [2].

3.2 Funcionamento da Cache e Threading na Conexão dos Componentes da Rede

Com a Cache, os switchs da rede passam a enxergar as Caches como Controladores, e enviam PacketIn diretamente para a Cache. Em cada Cache há um socket em modo listening, escutando na porta que os Switches acreditam ser do Controlador. A cada conexão recebida pela Cache, um socket é associado à essa conexão (entre Switch e Cache, aqui denominado socket1) e um outro socket é criado e conectado na porta real do Controlador (entre Cache e Controlador, aqui denominado socket2). Isto é representado na Fig. 3.4. Tendo esses dois sockets, são criadas duas threads. A primeira thread fica escutando no socket1, recebendo as mensagens que vêm do Switch e verificando se alguma regra de Cache é aplicável ao pacote recebido. Caso seja aplicável, ela mesmo responde ao socket1 com um pacote Flow Mod pertinente. Caso contrário, redireciona o pacote ao socket2, fazendo com que ele vá ao Controlador. A segunda thread fica escutando no socket2, recebendo as mensagens que vêm do Controlador, gravando-as em Cache e redirecionando-as para o socket1, fazendo com que cheguem ao Switch. Desse modo, a existência ou não de uma Cache fica transparente aos Switches que participam da rede.

A Cache criada para armazenamento das regras de Flow é uma TTL, com tamanho de 10 slots, por Cache, e 20 segundos de tempo de vida, implementada pela biblioteca **cachetools**. A biblioteca usada para criar as threads (**threading**) já tem mecanismos prontos para evitar que as threads escrevam ao mesmo tempo nos slots de memória da Cache, garantindo dessa forma a integridade dos dados armazenados. Dentre os mecanismos da biblioteca, utilizamos **conditions**, que permitem a realização de um sistema de tranca em recursos, garantindo acesso único simultâneo à Cache.

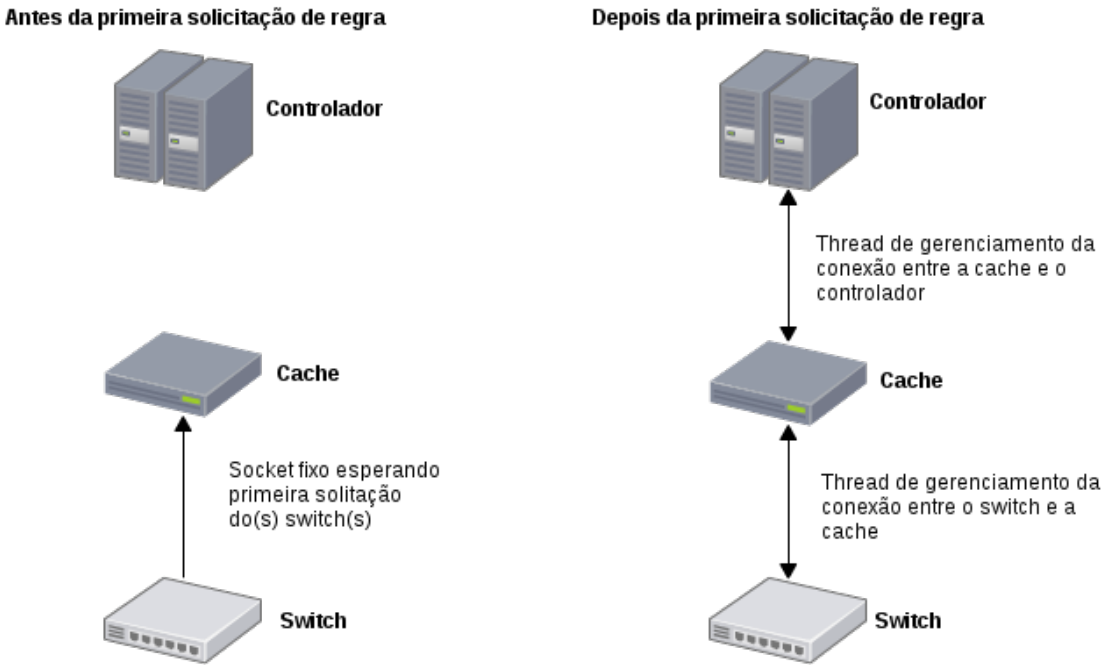

Figura 3.4: Cache e Threads

Sem a implementação da Cache, quando um Switch recebe um pacote que não possui regra instalada em sua Flow Table, é enviada uma requisição direto ao Controlador que, após avaliar o pacote, responde diretamente para o Switch. Com a implementação da Cache, quando ocorrer a mesma situação, a mensagem de solicitação de regra é enviada primeiramente para a Cache conectada ao Switch, onde é feita uma busca da informação da regra com base no endereço de destino, com um funcionamento de chave/valor, ou seja, dado o endereço de IP de destino obtém-se o pacote com a regra solicitada. Caso a informação esteja na Cache e tenha o tempo de vida válido, é realizado o retorno do pacote com a instalação da regra para o Switch solicitante. Caso a informação não esteja na Cache, seja por nunca ter sido solicitada e armazenada ou por expiração do tempo de vida, a mensagem é redirecionada para o Controlador sem alteração dos dados e lá é realizado o mesmo procedimento descrito acima onde não é implementada a Cache, porém, ao retornar o pacote, o Controlador envia o pacote para a Cache que, baseada no novo endereço de destino, armazenará os dados que correspondem a inserção de novos Flows, podendo assim responder novas solicitações que contenham o novo endereço de destino salvo na Cache. Após salvar os dados, a Cache encaminha o pacote para o Switch.

3.3 Resultados

Para realizar testes de tempo de resposta e número de pacotes enviado ao Controlador, a aplicação escolhida foi um Firewall simples. O Firewall implementado no Controlador envia regras de *drop* de pacotes com base no IP de destino. Por exemplo, caso um Switch não possua uma regra para pacote com IP destino X, envia um PacketIn para o Controlador, que analisa se o IP destino X consta nos IP's bloqueados. Caso não possua, envia um Flow Mod com a instalação da regra normalmente. Caso possua, envia um Flow Mod para instalação porém nesse pacote não contém dados de *Action*, o que significa que qualquer enviado para o IP X será descartado pelo Switch, enquanto a regra na Flow Table for válida.

A metodologia utilizada para os experimentos consiste em realizar testes com o comando *pingall*, nativo do Mininet, em duas topologias diferentes em uma rede que utiliza a aplicação de Firewall no Controlador SDN, sendo esses testes realizados de maneira comparativa com e sem a implementação de Cache. Para cada combinação de topologia e componentes (com ou sem Cache) foram realizados três testes e, com os resultados, foi realizado o cálculo da média simples para determinar o Tempo Médio de Resposta (TMR), medido em segundos, entre o Controlador e o Switch e o número de requisições que chegam ao Controlador durante sua execução.

As topologias que fizeram parte dos experimentos foram as seguintes:

- **Topologia 1 (TP1)**: Uma rede com um controlador SDN, quatro Switches, oito Hosts (2 por Swicth) e 30ms de latência entre os Switches e o Controlador.
- **Topologia 2 (TP2)**: Uma rede com um Controlador SDN, duas Caches, quatro Switches (2 por Cache), oito Hosts (2 por Switch) e 30ms de latência entre as Caches e o Controlador.

- **Topologia 3 (TP3)**: Uma rede com um Controlador SDN, nove Switches, vinte e sete Hosts (3 por Switch) e 30ms de latência entre os Switches e o Controlador.
- **Topologia 4 (TP4)**: Uma rede com um Controlador SDN, três Caches, nove Switches (3 por Cache), vinte e sete Hosts (3 por Switch) e 30ms de latência entre as Caches e o Controlador.

Processador	AMD FX(tm)-6300 Six-Core Processor × 6 - 3.5GHz - 8mb Cache
Memória	8GB - 1866MHz
HD	500GB
OS	64bit Ubuntu
Placa Mãe	Gigabyte GA-78LMT-S2, chipset 760G, AMD AM3+ - BOX

Tabela 3.1: Especificações da máquina utilizada nos experimentos

Todos os testes descritos abaixo foram realizados utilizando o computador cujas especificações encontram-se na Tabela 3.1.

- **Teste 1 (T1)**: Execução do comando *pingall* nas topologias TP1 e TP2, com o Firewall configurado para bloquear os seguintes endereços: 10.0.0.2 e 10.0.0.5. Os resultados da execução dos teste na TP1 e na TP2 encontram-se nas tabelas 3.2 e 3.3, respectivamente. A tabela 3.2 mostra uma média de 5.58 segundos de TMR, enquanto na tabela 3.3 podemos observar um TMR de 1.13 segundos, o que representa um ganho no tempo de resposta de até 5 vezes. Podemos notar também que o número de PacketIn's que chegam ao Controlador é menos da metade nos testes executados na topologia com Cache, uma vez que a utilização da mesma diminuiu a necessidade de requisições ao Controlador por parte dos Switches.

Execução	**TMR**	**Packetin**
1ª	4.19	22
2ª	6.29	22
3ª	5.27	22
Média:	5.58	22

Tabela 3.2: Execução do teste T1 na topologia TP1 (sem cache)

Execução	Cache	TMR	Packetin
1ª	**C1**	0.55	10
1ª	**C2**	1.80	
2ª	**C1**	0.83	12
2ª	**C2**	1.63	
3ª	**C1**	0.55	10
3ª	**C2**	1.43	
Média C1		0.64	10.6
Média C2		1.62	
Média Geral		1.13	

Tabela 3.3: Execução do teste T1 na topologia TP2 (com cache)

- **Teste 2 (T2)**: Execução do comando *pingall* nas topologias TP1 e TP2, com o Firewall configurado para bloquear os seguintes endereços: 10.0.0.3 e 10.0.0.4. Os resultados da execução dos teste na TP1 e na TP2 encontram-se nas tabelas 3.4 e 3.5, respectivamente. A tabela 3.4 mostra uma média de 4.25 segundos de TMR, enquanto na tabela 3.5 podemos observar um TMR de 1.16 segundos, o que representa um ganho no tempo de resposta de até 4 vezes. Podemos notar também que o número de PacketIn's que chegam ao Controlador é pouco mais da metade nos testes executados na topologia com Cache, uma vez que a utilização da mesma diminuiu a necessidade de requisições ao Controlador por parte dos Switches.

Execução	TMR	Packetin
1ª	3.63	20
2ª	4.40	20
3ª	4.73	20
Média:	4.25	20

Tabela 3.4: Execução do teste T2 na topologia TP1 (sem cache)

Execução	Cache	TMR	Packetin
1ª	**C1**	1.11	11
1ª	**C2**	0.88	
2ª	**C1**	0.87	12
2ª	**C2**	1.70	
3ª	**C1**	1.15	11
3ª	**C2**	1.30	
Média C1		1.04	11.3
Média C2		1.29	
Média Geral		1.16	

Tabela 3.5: Execução do teste T2 na topologia TP2 (com cache)

- **Teste 3 (T3)**: Execução do comando *pingall* nas topologias TP3 e TP4, com o Firewall configurado para bloquear os seguintes endereços: 10.0.0.2 e 10.0.0.5. Os resultados da execução dos teste na TP3 e na TP4 encontram-se nas tabelas 3.6 e 3.7, respectivamente. A tabela 3.6 mostra uma média de 31.8 segundos de TMR, enquanto na tabela 3.7 podemos observar um TMR de 15.79 segundos, o que representa um ganho no tempo de resposta de até 2 vezes. Podemos notar também que o número de PacketIn's que chegam ao Controlador é 21.5% menor testes executados na topologia com Cache, uma vez que a utilização da mesma diminuiu a necessidade de requisições ao Controlador por parte dos Switchs. Vale ressaltar que houve uma queda na diminuição de PacketIn's entre as topologias na execução desse teste quando comparado aos testes T1 e T2.

Execução	TMR	Packetin
1ª	32.16	72
2ª	31.58	71
3ª	31.68	71
Média:	31.80	71.3

Tabela 3.6: Execução do teste T3 na topologia TP3 (sem cache)

Execução	Cache	TMR	Packetin
1ª	**C1**	17.39	56
1ª	**C2**	7.65	
1ª	**C3**	18.47	
2ª	**C1**	17.94	56
2ª	**C2**	7.62	
2ª	**C3**	21.54	
3ª	**C1**	15.45	56
3ª	**C2**	7.75	
3ª	**C3**	21.77	
Média C1		19.13	56
Média C2		7.67	
Média C3		20.59	
Média Total		15.79	

Tabela 3.7: Execução do teste T3 na topologia TP4 (com cache)

- **Teste 4 (T4)**: Execução do comando *pingall* nas topologias TP3 e TP4, com o Firewall configurado para bloquear os seguintes endereços: 10.0.0.3, 10.0.0.4, 10.0.0.8, 10.0.0.15 e 10.0.0.20. Os resultados da execução dos teste na TP3 e na TP4 encontram-se nas tabelas 3.8 e 3.9, respectivamente. A tabela 3.8 mostra uma média de 53.35 segundos de TMR, enquanto na tabela 3.9 podemos observar um TMR de 26.32 segundos, o que representa um ganho no tempo de resposta de até 2 vezes. Podemos notar também que o número de PacketIn's que chegam ao Controlador é cerca de 12.1% menor testes executados na topologia com Cache, uma vez que a utilização da mesma diminuiu a necessidade de requisições ao Controlador por parte dos Switches. Vale ressaltar que houve uma queda na diminuição de PacketIn's entre as topologias na execução desse teste quando comparado aos testes T1 e T2.

Execução	TMR	Packetin
1ª	54.51	183
2ª	52.73	182
3ª	52.82	182
Média:	53.35	182

Tabela 3.8: Execução do teste T4 na topologia TP3 (sem cache)

Execução	Cache	TMR	Packetin
1ª	**C1**	23.80	160
1ª	**C2**	15.61	
1ª	**C3**	37.39	
2ª	**C1**	30.08	160
2ª	**C2**	15.27	
2ª	**C3**	37.92	
3ª	**C1**	27.44	160
3ª	**C2**	14.63	
3ª	**C3**	34.81	
Média C1		27.10	160
Média C2		15.17	
Média C3		36.70	
Média Total		26.32	

Tabela 3.9: Execução do teste T4 na topologia TP4 (com cache)

Nota-se que as topologias testadas tiveram resultados com diferentes tempos de reposta nas Caches. Presume-se que as diferenças de resultados foram afetadas diretamente pelo Tempo de vida das entradas de cache (TTL), pela rede inteira executada no mesmo computador e compartilhando recursos, além da latência gerada a cada nova solicitação ao Controlador.

Após a execução dos testes, obtivemos os resultados em porcentagem da diminuição no TMR e da diminuição no número de requisições ao Controlador, nos testes onde a Cache está implementada. Os resultados obtidos, em porcentagem, encontram-se na tabela 3.10. Com isso pudemos notar que, com o aumento da topologia, existe um menor ganho de desempenho na utilização da Cache em comparação a mesma topologia sem a utilização de Caches. Essa perda de desempenho com o crescimento da topologia deve-se ao fato de que os testes foram realizados em uma só máquina e, sendo assim, o crescimento pode gerar atrasos por falta de recursos da máquina. Além disso, devido a característica sequencial da execução do comando *pingall* o aumento da topologia impacta no desempenho dos testes de forma negativa, uma vez que cada Host deve enviar um ping para todos os outros Hosts daquela topologia enquanto os outros Hosts aguardam sua vez de enviar o comando ping, desse modo, com o aumento do tempo para que cada Host comece a enviar o comando ping, os dados da Cache vão se tornando inválidos, o que leva a um número maior de solicitações ao Controlador e um maior tempo de resposta.

Teste:	Diminuição no TMR (%)	Diminuição no número de requisições ao Controlador (%)
T1	79%	54%
T2	72%	45%
T3	50%	21%
T4	51%	12%

Tabela 3.10: Resultado dos testes

Capítulo 4

Conclusão

SDN é um modelo emergente utilizado para projetar, criar e gerenciar redes de computadores, e tem como objetivo superar as limitações de infra-estrutura das redes convencionais. Com a separação do controle da rede e o plano de encaminhamento tornou-se possível que o controle da rede se tornasse diretamente programável e que a infraestrutura subjacente pudesse ser abstraída para aplicações e serviços de rede. Desse modo, SDN traz como benefícios agilidade e flexibilidade no desenvolvimento e administração de uma rede, tornando-se um modelo que permite um maior grau de inovação.

O foco principal das redes definidas por software são as aplicações realizadas no Controlador, e implementam serviços oferecidos na rede. O Controlador representa o cérebro da rede, já que as decisões quanto ao funcionamento da rede são de sua responsabilidade. Um dos grandes problemas encontrados no SDN é a escalabilidade da rede, visto que quanto maior a rede se torna, maior será a ocorrência de eventos e requisições enviadas ao Controlador. Sendo assim, em uma rede que possui um único Controlador designado para controlar toda a a funcionalidade da rede o Controlador pode não conseguir responder todas as requisições por sobrecarga de serviço a medida que a rede aumenta.

Para intermediar a comunicação entre o plano de dados e o plano de controle de uma SDN é necessário um protocolo de comunicação, atualmente o protocolo mais utilizado para esse fim é o OpenFlow, que consiste em três componentes: tabelas de fluxo, que contém ações para cada registro que case com suas regras; um canal seguro, que faz a conexão entre o Switch e o Controlador, permitindo envio de comandos e pacotes entre as partes; e por último, o próprio protocolo responsável pela comunicação entre os componentes.

Para esse trabalho utilizamos o Mininet, um emulador de redes de computadores que contém suporte à SDN, mais especificamente por meio de emulações de uma rede Openflow. O Mininet conta com facilitadores para identificar o comportamento do sistema, além de executar código nativo Unix/Linux e ter conexão com aplicativos de mensuração de desempenho, tal como o iperf. Assim, o código desenvolvido e testado nos Controladores, Switches e Hosts do Mininet, é facilmente adaptado aos dispositivos físicos, com pequenos ajustes nas configurações.

Com isso, propusemos nesse trabalho a implementação de uma Cache para as mensagens do Controlador SDN, visando reduzir o número de requisições ao mesmo. Para isso, utilizamos o POX, um Controlador SDN em Python que é recomendado pela comunidade de SDN's para a utilização no ensino e em pesquisas. A Cache foi implementada como um processo separado, intermediando a comunicação entre Switches e Controlador, permitindo ao Switch pensar que a Cache é um Controlador SDN.

As caches implementadas atuam como um agente intermediário entre os Switches e o Controlador SDN da rede. Uma vez que o Switch requer uma nova regra, para um Flow que não está presente em sua Flow Table, é gerada uma mensagem de solicitação e enviada para a uma das Caches. Assim que uma mensagem de solicitação de regra chega na Cache é feita uma tentativa de match na tabela da Cache e, caso encontre, é retornada a mensagem ao Switch contendo a regra solicitada. Caso não encontre, a mensagem é então redirecionada ao Controlador SDN que, após gerar a regra e inseri-la na mensagem, devolve a mesma para a Cache, que por sua vez atualiza sua tabela com a informação e um TTL pré-definido e retorna a mensagem para o Switch.

Ao realizar os testes com diversas topologias diferentes, concluímos que a utilização de Cache das mensagens do Controlador SDN para aumentar a escalabilidade em uma rede com apenas um Controlador e vários Switches pode trazer uma melhora significativa no desempenho da rede quando o fluxo de requisições ao Controlador começa a aumentar, pois as Caches resultaram em uma diminuição de requisições ao Controlador e, por consequência, uma diminuição no tempo de resposta para as solicitações de regras de fluxo para os Switches. Porém, quanto maior torna-se a topologia, menor é o ganho de desempenho com a utilização da solução de Caches. Como discutido neste trabalho essa perda deve-se ao fato de que os testes foram realizados em um só computador e, sendo assim, o crescimento pode gerar atrasos por falta de recursos da máquina. Além disso, devido a característica sequencial da execução do comando *pingall* o aumento da topologia impacta no desempenho dos testes de forma negativa, uma vez que cada Host deve enviar um ping para todos os outros Hosts daquela topologia enquanto os outros Hosts aguardam sua vez de enviar o comando ping, desse modo, com o aumento do tempo para que cada Host comece a enviar o comando ping, os dados da Cache vão se tornando inválidos, o que leva a um número maior de solicitações ao Controlador e um maior tempo de resposta.

Além disso, a solução que propomos permite que o Controlador SDN mantenha sua visão geral da topologia da rede, ao contrário do que acontece quando adicionamos mais Controladores a rede para tentar solucionar esse mesmo problema. Ainda sobre isso, os Controladores possuem uma lógica muito mais complexa do que uma Cache e, sendo assim, torna a utilização de múltiplos Controladores mais lenta, mesmo que minimamente, do que a utilização de Caches.

Referências Bibliográficas

[1] OpenFlow 1.0. Especification. http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf, 2016.

[2] Flowgrammable Actions. Actions - openflow. http://flowgrammable.org/sdn/openflow/actions/, 2016.

[3] Ali Al-Shabibi. Pox wiki. https://openflow.stanford.edu/display/ONL/POX+Wiki, 2012.

[4] cachetools. cachetools. https://pythonhosted.org/cachetools/#cache-implementations, 2016.

[5] POX Controller. Pox controller - official website. http://www.noxrepo.org/pox/about-pox/, 2016.

[6] ETSI. Network functions virtualisation (nfv); architectural framework. http://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf, 2013. ETSI - European Telecommunications Standards Institute.

[7] Flowgrammable. Message layer - packetin. http://flowgrammable.org/sdn/openflow/message-layer/packetin/, 2016.

[8] Anmol Garg. *Network Function Virtualization*. PhD thesis, Indian Institute of Technology Bombay, 2014.

[9] Masayoshi Kobayashi, Srini Seetharaman, Guru Parulkar, Guido Appenzeller, Joseph Little, Johan Van Reijendam, Paul Weissmann, and Nick McKeown. Maturing of openflow and software-defined networking through deployments. *Computer Networks*, 61:151–175, 2014.

[10] Diogo MF Mattos, Martin Andreoni Lopez, Lyno Henrique G Ferraz, Duarte, and Otto Carlos MB. Controlador resiliente com distribuiçao eficiente para redes definidas por software. 2015. Rio de Janeiro, Brazil.

[11] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *Newsletter*, 38:69–74, March 2008.

[12] Mininet. Introduction to mininet - official website. `https://github.com/mininet/mininet/wiki/Introduction-to-Mininet`, 2016.

[13] A. J. Pinheiro. Escalabilidade do plano de controle em redes openflow, 2013. Universidade Federal do Ceará - UFC.

[14] Sdxcentral. What is nfv – network functions virtualization – definition? `https://www.sdxcentral.com/nfv/definitions/whats-network-functions-virtualization-nfv/`, 2012-2016.

[15] Open vSwitch. Open vswitch - official website. `http://openvswitch.org/`, 2014.

[16] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali. On scalability of software-defined networking. *IEEE Communications Magazine*, 51(2):136–141, February 2013.